Đã bắt đầu vào lúc	Thứ bảy, 19 Tháng mười một 2022, 9:37 AM
Tình trạng	Đã hoàn thành
Hoàn thành vào lúc	Thứ bảy, 19 Tháng mười một 2022, 10:17 AM
Thời gian thực hiện	39 phút 34 giây
Điểm	3,00/3,00
Điểm	**10,00** của 10,00 (**100**%)

Câu hỏi 1

Chính xác

Điểm 1,00 của 1,00

In this question, you have to perform **add** on AVL tree. Note that:

- When adding a node which has the same value as parent node, add it in the **right sub tree**.

Your task is to implement function: **insert**. You could define one or more functions to achieve this task.

```
#include <iostream>
#include <math.h>
#include <queue>
using namespace std;
#define SEPARATOR "#<ab@17943918#@>#"

enum BalanceValue
{
    LH = -1,
    EH = 0,
    RH = 1
};

void printNSpace(int n)
{
    for (int i = 0; i < n - 1; i++)
        cout << " ";
}

void printInteger(int &n)
{
    cout << n << " ";
}

template<class T>
class AVLTree
{
public:
    class Node;
private:
    Node *root;
protected:
    int getHeightRec(Node *node)
    {
        if (node == NULL)
            return 0;
        int lh = this->getHeightRec(node->pLeft);
        int rh = this->getHeightRec(node->pRight);
        return (lh > rh ? lh : rh) + 1;
    }
public:
    AVLTree() : root(nullptr) {}
    ~AVLTree(){}
    int getHeight()
    {
        return this->getHeightRec(this->root);
    }
    void printTreeStructure()
    {
        int height = this->getHeight();
        if (this->root == NULL)
        {
            cout << "NULL\n";
            return;
        }
        queue<Node *> q;
        q.push(root);
        Node *temp;
        int count = 0;
        int maxNode = 1;
        int level = 0;
        int space = pow(2, height);
        printNSpace(space / 2);
        while (!q.empty())
        {
            temp = q.front();
            q.pop();
            if (temp == NULL)
            {
```

```
                cout << " ";
                q.push(NULL);
                q.push(NULL);
            }
            else
            {
                cout << temp->data;
                q.push(temp->pLeft);
                q.push(temp->pRight);
            }
            printNSpace(space);
            count++;
            if (count == maxNode)
            {
                cout << endl;
                count = 0;
                maxNode *= 2;
                level++;
                space /= 2;
                printNSpace(space / 2);
            }
            if (level == height)
                return;
        }
    }

    void insert(const T &value)
    {
        //TODO
    }

    class Node
    {
    private:
        T data;
        Node *pLeft, *pRight;
        BalanceValue balance;
        friend class AVLTree<T>;

    public:
        Node(T value) : data(value), pLeft(NULL), pRight(NULL), balance(EH) {}
        ~Node() {}
    };
};
```


For example:

Test	Result

AVLTree<int> avl;
for (int i = 0; i < 9; i++){
avl.insert(i);
}
avl.printTreeStructure();
	3
	1 5
	0 2 4 7
	6 8

AVLTree<int> avl;
for (int i = 10; i >= 0; i--){
avl.insert(i);
}
avl.printTreeStructure();
	7
	3 9
	1 5 8 10
	0 2 4 6

Answer: (penalty regime: 0 %)

Reset answer

```
1  //Helping functions
2  Node* rotatepRight(Node *&node)
3  {
4      Node* tmp = node->pLeft;
```

```
    node->pLeft = tmp->pRight;
    tmp->pRight = node;
    return tmp;
}
Node* rotatepLeft(Node *&node)
{
    Node* tmp = node->pRight;
    node->pRight = tmp->pLeft;
    tmp->pLeft = node;
    return tmp;

}

Node* pRightBalance(Node *&node, bool &taller)
{
     if (node->pRight->balance == RH) {
                node->balance = EH;
                node->pRight->balance = EH;
                node = rotatepLeft(node);
                taller = false;
                return node;
            }
            //lor
            Node* pLeftSubTree = node->pRight->pLeft;
            if (pLeftSubTree->balance == LH) {
                node->balance = EH;
                node->pRight->balance = RH;
            }
            else if (pLeftSubTree->balance == EH) {
                node->balance = EH;
                node->pRight->balance = EH;
            }
            else {
                node->balance = LH;
                node->pRight->balance = EH;
            }
            pLeftSubTree->balance = EH;
            node->pRight = rotatepRight(node->pRight);
            node = rotatepLeft(node);
            taller = false;
            return node;
}


Node* pLeftBalance(Node *&node, bool &taller)
{
    //lol
            if (node->pLeft->balance == LH) {
                node->balance = EH;
                node->pLeft->balance = EH;
                node = rotatepRight(node);
                taller = false;
                return node;
            }
            //rol
            Node* pRightSubTree = node->pLeft->pRight;
            if (pRightSubTree->balance == LH) {
                node->balance = RH;
```


	Test	Expected	Got	

✔	`AVLTree<int> avl;`
	`for (int i = 0; i < 9; i++){`
	`  avl.insert(i);`
	`}`
	`avl.printTreeStructure();`
		3
		1 5
		0 2 4 7
		6 8
			3
			1 5
			0 2 4 7
			6 8
				✔

	Test	Expected	Got	

✔	`AVLTree<int> avl;`
	`for (int i = 10; i >= 0; i--){`
	`\tavl.insert(i);`
	`}`
	`avl.printTreeStructure();`
		`7`
		`3 9`
		`1 5 8 10`
		`0 2 4 6`
			`7`
			`3 9`
			`1 5 8 10`
			`0 2 4 6`
				✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 2

Chính xác

Điểm 1,00 của 1,00

In this question, you have to perform **delete on AVL tree**. Note that:

- Provided **insert** function already.

Your task is to implement two functions: **remove**. You could define one or more functions to achieve this task.

```
#include <iostream>
#include <math.h>
#include <queue>
using namespace std;
#define SEPARATOR "#<ab@17943918#@>#"

enum BalanceValue
{
    LH = -1,
    EH = 0,
    RH = 1
};

void printNSpace(int n)
{
    for (int i = 0; i < n - 1; i++)
        cout << " ";
}

void printInteger(int &n)
{
    cout << n << " ";
}

template<class T>
class AVLTree
{
public:
    class Node;
private:
    Node *root;
protected:
    int getHeightRec(Node *node)
    {
        if (node == NULL)
            return 0;
        int lh = this->getHeightRec(node->pLeft);
        int rh = this->getHeightRec(node->pRight);
        return (lh > rh ? lh : rh) + 1;
    }
public:
    AVLTree() : root(nullptr) {}
    ~AVLTree(){}
    int getHeight()
    {
        return this->getHeightRec(this->root);
    }
    void printTreeStructure()
    {
        int height = this->getHeight();
        if (this->root == NULL)
        {
            cout << "NULL\n";
            return;
        }
        queue<Node *> q;
        q.push(root);
        Node *temp;
        int count = 0;
        int maxNode = 1;
        int level = 0;
        int space = pow(2, height);
        printNSpace(space / 2);
        while (!q.empty())
        {
            temp = q.front();
            q.pop();
            if (temp == NULL)
            {
```

```
                cout << " ";
                q.push(NULL);
                q.push(NULL);
            }
            else
            {
                cout << temp->data;
                q.push(temp->pLeft);
                q.push(temp->pRight);
            }
            printNSpace(space);
            count++;
            if (count == maxNode)
            {
                cout << endl;
                count = 0;
                maxNode *= 2;
                level++;
                space /= 2;
                printNSpace(space / 2);
            }
            if (level == height)
                return;
        }
    }

    void remove(const T &value)
    {
        //TODO
    }

    class Node
    {
    private:
        T data;
        Node *pLeft, *pRight;
        BalanceValue balance;
        friend class AVLTree<T>;

    public:
        Node(T value) : data(value), pLeft(NULL), pRight(NULL), balance(EH) {}
        ~Node() {}
    };
};
```


For example:

Test	Result

AVLTree<int> avl;
int arr[] = {10,52,98,32,68,92,40,13,42,63};
for (int i = 0; i < 10; i++){
avl.insert(arr[i]);
}
avl.remove(10);
avl.printTreeStructure();
	52
	32 92
	13 40 68 98
	42 63

AVLTree<int> avl;
int arr[] = {10,52,98,32,68,92,40,13,42,63,99,100};
for (int i = 0; i < 12; i++){
avl.insert(arr[i]);
}
avl.remove(13);
avl.printTreeStructure();
	52
	32 92
	10 40 68 99
	42 63 98 100

Answer: (penalty regime: 0 %)

Reset answer

```
//Helping functions
Node *rotatepRight(Node *&node)
{
    Node* tmp = node->pLeft;
    node->pLeft = tmp->pRight;
    tmp->pRight = node;
    return tmp;
}


Node *rotatepLeft(Node *&node)
{
    Node* tmp = node->pRight;
    node->pRight = tmp->pLeft;
    tmp->pLeft = node;
    return tmp;

}



Node *removepRightBalance(Node *&node, bool &shorter)
{
    if (node->balance == LH) {
            node->balance = EH;
            return node;
        }
        if (node->balance == EH) {
            node->balance = RH;
            shorter = false;
            return node;
        }
        if (node->balance == RH) {
            //lor
            if (node->pRight->balance == LH) {
                Node* pLeftOfpRightTree = node->pRight->pLeft;
                if (pLeftOfpRightTree->balance == LH) {
                    node->balance = EH;
                    node->pRight->balance = RH;
                }
                else if (pLeftOfpRightTree->balance == EH) {
                    node->balance = EH;
                    node->pRight->balance = EH;
                }
                else {
                    node->balance = LH;
                    node->pRight->balance = EH;
                }
                pLeftOfpRightTree->balance = EH;
                node->pRight = rotatepRight(node->pRight);
                node = rotatepLeft(node);
                //shorter = false;
            }
            else if (node->pRight->balance == RH) {
                node->balance = EH;
                node->pRight->balance = EH;
                node = rotatepLeft(node);
                //shorter = false;
            }
            else {
                node->balance = RH;
                node->pRight->balance = LH;
```

	Test	Expected	Got	

✔	AVLTree<int> avl;
	int arr[] = {10,52,98,32,68,92,40,13,42,63};
	for (int i = 0; i < 10; i++){
	\tavl.insert(arr[i]);
	}
	avl.remove(10);
	avl.printTreeStructure();
		52
		32 92
		13 40 68 98
		42 63
			52
			32 92
			13 40 68 98
			42 63
				✔

✔	AVLTree<int> avl;
	int arr[] = {10,52,98,32,68,92,40,13,42,63,99,100};
	for (int i = 0; i < 12; i++){
	\tavl.insert(arr[i]);
	}
	avl.remove(13);
	avl.printTreeStructure();
		52
		32 92
		10 40 68 99
		42 63 98 100
			52
			32 92
			10 40 68 99
			42 63 98 100
				✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 3

Chính xác

Điểm 1,00 của 1,00

In this question, you have to search and print inorder on **AVL tree**. You have o implement functions: **search** and **printInorder** to complete the task. Note that:

- When the tree is null, don't print anything.

- There's a whitespace at the end when print the tree inorder in case the tree is not null.

- When tree contains value, search return true.

```
#include <iostream>
#include <queue>
using namespace std;
#define SEPARATOR "#<ab@17943918#@>#"

enum BalanceValue
{
    LH = -1,
    EH = 0,
    RH = 1
};

template<class T>
class AVLTree
{
public:
    class Node;
private:
    Node *root;
public:
    AVLTree() : root(nullptr) {}
    ~AVLTree(){}

    void printInorder(){
        //TODO
    }

    bool search(const T &value){
        //TODO
    }

    class Node
    {
    private:
        T data;
        Node *pLeft, *pRight;
        BalanceValue balance;
        friend class AVLTree<T>;

    public:
        Node(T value) : data(value), pLeft(NULL), pRight(NULL), balance(EH) {}
        ~Node() {}
    };
};
```


For example:

Test	Result

AVLTree<int> avl;
int arr[] = {10,52,98,32,68,92,40,13,42,63,99,100};
for (int i = 0; i < 12; i++){
avl.insert(arr[i]);
}
avl.printInorder();
cout << endl;
cout << avl.search(10);
	10 13 32 40 42 52 63 68 92 98 99 100
	1

Answer: (penalty regime: 0 %)

```
void printInorderRec(Node* node){
    if(node == nullptr) return;
    if(node->pLeft && node->pRight){
        printInorderRec(node->pLeft);
        cout<<" "<< node->data<<" ";
        printInorderRec(node->pRight);
    }
    else if(node->pRight){
        cout<< node->data<<" ";
        printInorderRec(node->pRight);
    }
    else if(node->pLeft){
        printInorderRec(node->pLeft);
        cout<<" "<<node->data;
    }
    else cout<< node->data;

}
void printInorder(){
    printInorderRec(root);
}

bool search(const T &value){
    Node* tmp  = root;
    while(tmp){
        if(tmp->data == value) return true;
        else{
            if(tmp->data < value) tmp = tmp->pRight;
            else tmp = tmp->pLeft;
        }
    }
    return false;
}
```

	Test	Expected	Got	

✔	AVLTree<int> avl;
	int arr[] =
	{10,52,98,32,68,92,40,13,42,63,99,100};
	for (int i = 0; i < 12; i++){
	\tavl.insert(arr[i]);
	}
	avl.printInorder();
	cout << endl;
	cout << avl.search(10);
		10 13 32 40 42 52 63 68 92 98 99 100
		1
			10 13 32 40 42 52 63 68 92 98 99 100
			1
				✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.